유연하고 확장 가능한 CBD 어플리케이션 프레임워크와 성능분석

(A Flexbile and Extensible CBD Application Framework and Performance Analysis)

이 용 환 †

(Yong Hwan Lee)

요 약 최근 많은 소프트웨어 개발 프로젝트들은 지속적으로 요구사항들이 변경되는 대규모의 복잡한 시스템이며 컴포넌트 기반이나 제품계열 소프트웨어 개발 방식과 같이 아키텍처 기반으로 소프트웨어를 개발하고 있다. 이러한 소프트웨어 개발환경에서 생산성이나 유지보수 향상을 위해서는 무엇보다도 유연하고 확장 가능한 아키텍처가 필요하다. 본 논문에서는 CBD(Component Base Development) 기반의 비즈니스 어플리케이션 개발 및 유지보수 시 생산성과 유지보수 향상을 위해 필요한 유연성과 확장성 품질속성을 가진 소프트웨어 아키텍처를 제시하고 이들 품질속성을 달성하기 위한 방법들을 아키텍처 모델을 통해 설명한다. 또한, 본 논문에서는 최근 프리젠테이션(Presentation) 프레임워크(Framework)로 잘 알려진 Struts와 JBean이라 불리는 본 논문에서 제안한 프레임워크의 성능을 비교했다. JBean프레임워크는 프리젠테이션 프레임워크와 비즈니스 로직 프레임워크를 가지고 있기 때문에 공정한 실험을 위해 JBean에서 비즈니스 로직 프레임워크에 관련된 모듈을 제거하고 단지 프리젠테이션단 프레임워크만 가지고 실험을 한 결과 JBean의 프리젠테이션 단 프레임워크가 평균 초당 18개의 거래를 더 처리하고 있다.

키워드 : 프레임워크, 성능분석, CBD, Struts, 어플리케이션 프레임워크

Abstract Current many software development projects gradually develop large volume of complex and changeable system. Architecture driven methodology is used for developing softwares like CBD (Component Based Development) or PLE (Product Line Engineering). To improve the development and maintainability in such an environment, the first of all needs the flexible and extensible architecture. This paper suggests the CBD application framework that has the flexbile and extensible architecture for improving development productivity and maintainability. In this paper, we also show basic performances of our suggested framework, comparing to a well-known web application framework, Struts.

Key words : Framework, Performance Analysis, CBD, Struts, Application framework

1. 서 론

소프트웨어 산업이 급속하고 다양한 형태로 발전해 가고 기업간 경쟁이 더욱 심화되어가면서 소프트웨어 재 사용성, 적시성 그리고 유지 보수성이 중요한 요인으로 대두되고 있으며[1,2] 이에 대한 솔루션으로 컴포넌트 기반[3]이나 제품계열 소프트웨어 개발(Product Line Software Development) 방식[4]들이 점차적으로 각광을 받고 있다. 이들 방식들은 표준 컴포넌트 기술을 이용하여 개발된 컴포넌트를 획득하여 조립하는 방식으로 아키텍처 기반으로 소프트웨어를 개발함으로서 높은 생산성과 품질을 제공하고 있다[5,6].

비즈니스 측면에서 볼 때 비즈니스 조직은 계속적으로 변경되는 비즈니스 요구사항 및 환경 그리고 프로세스 또는 워크플로우에 적절하게 대응해야 하며 새로운 비즈니스 요구에 신속하게 적응해야 한다[7,8]. 이러한 비즈니스 요구사항의 변경은 소프트웨어 개발과 유지보수를 매우 어렵게 하지만 비즈니스 목표와 환경이 변하기 때문에 일어날 수 밖에 없는 것이 현실이다. 따라서 아키텍처 중심적인 소프트웨어 개발 방식과 소프트웨어 시스템에 대한 지속적 요구사항 변화를 보이는 가변적인 상황에서 소프트웨어 개발 생산성이나 유지보수 향상을 위해서는 개발할 소프트웨어가 가지는 아키텍처는

† 정 회 원 : 건국대학교 컴퓨터학과 교수
yhlee@konkuk.ac.kr
논문접수 : 2005년 11월 25일
심사완료 : 2006년 9월 12일

무엇보다도 재구성을 쉽게 할 수 있을 정도로 유연하고 기능 추가를 용이하게 할 수 있어야 한다[9]. 즉, 아키텍처는 유연성과 확장성 품질속성을 만족해야 한다[6,10]. 이러한 아키텍처가 가지는 유연성과 확장성 품질 속성을 통해 개발 생산성을 향상하고 양질의 소프트웨어 품질을 제공하기 위해 반제품 형태의 어플리케이션 프레임워크 기반으로 소프트웨어를 개발하고 있다[11].

본 논문에서는 개발 업무가 많고 요구사항이 자주 변경되며 또한 각 업무 단위 별로 상충되는 아키텍처 상의 품질속성이나 설정들이 존재하는 개발 환경에서 생산성과 유지보수 향상을 위한 CBD 어플리케이션 프레임워크를 제시한다. 제시한 프레임워크는 생산성 향상을 위해 아키텍처 품질속성 중에 유연성과 확장성 품질속성을 가지고 있으며 프레임워크에 개발 툴킷이나 관리 툴킷을 가진 그레이박스 프레임워크(Graybox Framework)를 제공함으로서 개발 생산성을 향상시킬 수 있다.

아키텍처의 유연성과 확장성 품질 속성은 성능 품질 속성과 반비례 관계다. 가변성을 수용할 수 있는 유연한 아키텍처라 할지라도 성능 품질 속성이 만족되지 않으면 그 소프트웨에는 사용할 수 없을 것이다. 따라서 본 논문에서는 최근 프리젠테이션단 프레임워크로 잘 알려진 Struts와의 성능비교를 수행한 결과 JBean은 Struts보다 유연성과 확장성이 좋을 뿐만 아니라 평균 초당 18개의 거래를 더 처리할 수 있을 정도로 성능 품질이 우수하다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 제시한 프레임워크와 유사한 어플리케이션 프레임워크의 종류와 특성에 대해서 나열하고 프레임워크 기반 어플리케이션 개발시 개발 생산성 향상 방안에 대한 기존 연구를 기술한다. 3장에서는 본 논문에서 제시한 프레임워크의 전체적인 아키텍처를 제시하고 4장에서는 프레임워크의 유연성에 대해 5장에서는 확장성 달성방안에 대해 기술한다. 6장에서는 Struts 프레임워크와의 성능 비교 결과를 제시하고 마지막으로 6장에서는 결론을 기술한다.

2. 관련연구

2.1 VSSH 통합 프레임워크

90년대 후반 아파치 웹 서버와 톰캣, Resin 등 자바를 지원하는 오픈 소스 솔루션에 대한 우수함의 검증과 더불어 2000년대에 들어서 JUnit, CVS, Ant, Log4J와 같은 많은 오픈 소스 기술과 Struts[12], Turbine, Velocity[13], Spring[14]과 같은 공개된 오픈 소스 프레임워크가 일반화 될 정도로 오픈 소스 시장이 성숙해 왔다. 최근에 들어서는 국내에서 이러한 오픈 소스 프레임워크가 실제 J2EE 프로젝트에 적용되는 사례도 늘어가고 있으며 기업의 어플리케이션 제작 시 복잡도를 낮추기 위해 MVC 패턴이 녹아 있는 N 계층 C/S 환경을 구성하기 시작했으며 점차 시간이 흘러 가면서 대규모 웹 어플리케이션은 그림 1과 같은 다섯가지 계층으로 일반화 되어가고 있다.

최근 엔터프라이즈 어플리케이션 개발 프로젝트들에서는 각 층별 주요 프레임워크를 통합한 VSSH(Velocity, Struts, Spring, Hibernate) 통합 프레임워크를 고려하고 있다. 벨로시티는 프리젠테이션 단에서 데이타를 UI에 렌더링 하기 위한 프레임워크로서 개발자는 사용자에게 보여질 디자인을 위해 VTL을 통해 템플릿 파일을 작성한다. 그런 후에 이 템플릿 파일을 조정하기 위한 코드를 작성한다. VTL은 벨로시티에서 템플릿 개발에 사용되는 언어로서 레퍼렌스, 디렉티브 그리고 주석으로 구성된다.

그림 2는 클라이언트 요처에 대한 Struts 프레임워크의 흐름도이다. Struts는 프리젠테이션 단을 위한 프레임워크로서 MVC(Model-View-Controller) 아키텍처 패턴[15]을 가지며 이미 많은 사람들에 의해서 입증된 설계 프레임워크들을 기반으로 한 확장 가능한 웹 어플리케이션 프레임워크이다.

표 1은 Struts 프레임워크와 본 논문에서 제안한 프레임워크 이름인 JBean의 주요 모듈에 대한 비교이다. 클라이언트에서 요청이 들어온 두개의 프레임워크 전부 컨트롤(Control)에 해당하는 AS와 FS 서블릿이 처리한다. 두 프레임워크는 모두 모델에 해당하는 Action 클래스를 가지며 이 클래스의 훅 메소드에 비즈니스 로직을 코딩한다. 이러한 Action클래스를 로딩하는 역할을 하는 클래스가 Struts에서는 RequestProcessor에서 수행하며 JBean에서는 Action Processor가 수행한다. View

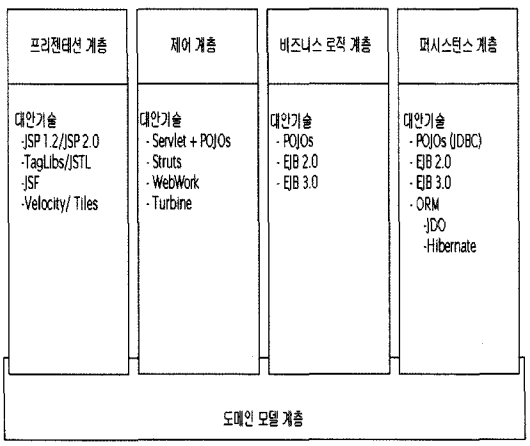

그림 1 비즈니스 시스템 아키텍처를 구성하는 5계층과 대안기술

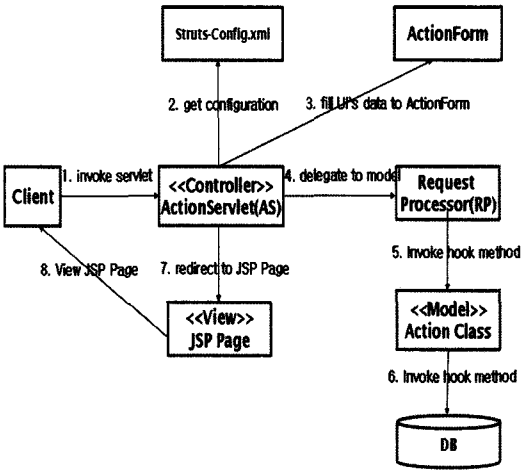

그림 2 Struts프레임워크 흐름도

표 1 Struts와 JBean의 주요 모듈 비교

	Struts	JBean
Control	ActionServlet(AS)	FrontServlet(FS)
Model	RequestProcessor(RP)와 Action	ActionProcessor(AP)와 Action
View	JSP	WebPageServlet(PS)와 JSP
UI 데이타 포함 객체	ActionForm	DTO(Data Transfer Object)

에서는 Struts와 JBean이 약간의 차이가 존재한다. Struts에서는 컨트롤인 AS에서 설정 파일 내용을 보고 바로 JSP로 이동하지만 JBean에서는 Composite View 패턴이나 View Helper패턴에 따라 별도의 페이지 구성을 담당하는 WebPageServlet인 PS을 거쳐 해당 JSP로 이동한다. 마지막으로 UI에서 전달된 객체를 담기 위해서 Struts는 ActionForm객체를 사용하지만 JBean에서는 DTO라는 객체를 사용해서 처리하고 있다.

여러 가지 면에서 Struts와 JBean 프레임워크는 유사하지만 Sturuts는 EJB[16]와 같은 비즈니스 로직 단을 커버하고 있지 않다. 스프링 프레임워크는 비즈니스 로직단 객체들을 관리하는 레이어 기반의 아키텍처 패턴을 가진 프레임워크이다. 이 프레임워크는 어플리케이션 개발에서 요구되는 인프라 서비스를 제공하며 테스트 기반 프로젝트에서 사용하기에 좋다. 하이버네이트는 객체와 관계 형데이타베이스 테이블간의 매핑을 지원하는 프레임워크로서 연관, 합성, 상속, 다형성 등과 같은 여러 관계를 지원한다. 또한 하이버네이트 쿼리 언어를 통한 강력한 쿼리 기능을 제공한다.

특정 계층에 한정된 프레임워크를 통합함으로써 엔터프라이즈 어플리케이션을 개발할 때는 각 프레임워크가 다른 프레임워크에 의존적이지 않으면서도 통합이 쉬어야 하며 유기적으로 연결될 수 있어야 한다. 이러한 통합에는 각 프레임워크가 가지는 아키텍처상의 구조에 대한 통합도 있지만 상이한 데이타 포맷 통합이나 그리고 개발이나 관리를 위한 툴킷 통합도 있다. VSSH와 같은 통합 어플리케이션은 약결합 형태의 통합을 추진하고 있지만 통합 용이성이나 개발이나 관리툴킷 통합 부재와 같은 많은 문제점을 가지고 있다.

다음으로 프레임워크 사용 편리성 차원에서 프레임워크는 프레임워크 내부 구조에 대한 이해를 가능한 최소화 시켜야 하며 사용법을 어렵게 해서는 안된다. 따라서 개발 경험 및 관련 기술에 대한 숙련도가 낮은 개발자도 최소한의 학습 및 사용 방법 숙지를 통해 어플리케이션을 구축할 수 있어야 한다. 위와 같은 점에서 Velocity, Struts, Spring과 같은 프레임워크는 복잡도를 사용자에게 직접 노출하여 확장 이용하게 함으로서 전체적인 시스템의 복잡도 및 유연성 측면에서 비효율적인 소프트웨어를 생산하게 된다.

2.2 프레임워크를 통한 개발 생산성 향상 방안

재사용 기법을 통해 개발 생산성을 향상시키기 위해서 컴포넌트 기반 어플리케이션 개발 프로젝트들은 프레임워크 기반으로 소프트웨어를 개발한다. 하지만 좀더 향상된 개발 생산성을 위해서는 그레이박스 형태의 프레임워크가 필요하며 또한 가변성을 수용할 수 있는 유연하고 확장 가능한 아키텍처가 필요하다[17]. 그림 3은 어플리케이션 프레임워크 개발 생산성 향상을 위한 그레이박스 프레임워크의 필요성에 대한 설명이다.

어플리케이션 프레임워크를 화이트 박스(Whitebox) 프레임워크라 한다면 이러한 프레임워크를 지원하는 툴킷(Toolkit)을 블랙박스 프레임워크(BlackBox Framework)라 한다. 최근 어플리케이션 프레임워크는 화이트 박스 프레임워크와 블랫박스 프레임워크를 동시에 가지는 그레이박스 프레임워크를 필요로 한다. 그레이박스 프레임워크는 코드생성과 관리운영에 대한 자동화를 지원함으로서 개발 생산성 및 유지보수를 용이하게 한다.

그레이박스 프레임워크 필요성에서 볼 때 Velocity, Struts, Spring, Hibernate와 같은 프레임워크는 각각 특정 단만을 관리하고 있지 웹, 비즈니스 로직, 레거시 단에 걸친 프레임워크가 아니며 개발 생산성이나 운영 유지보수를 위한 툴킷을 제공하지 있지 않다.

표 2는 Morisio가 프레임워크 사용 여부와 가변성에 따른 생산성을 비교한 실험 결과이다[18,19].

표 2의 MOD1과 MOD2는 업무 유형에 따라 분리한 것이고 LOC(Line Of Code)는 코드 라인을, LOC/Hour

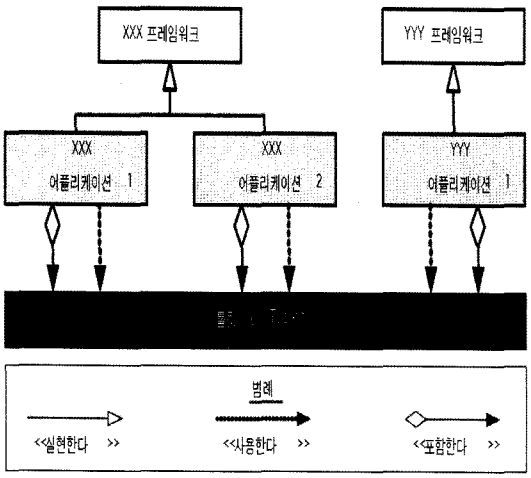

그림 3 개발 생산성 향상을 위한 그레이박스 프레임워크

표 2 프레임워크 사용 여부와 가변성에 따른 생산성 비교

Service	Variant	Framework	Language	Effort	LOC	Level	LOC/Hour
MOD1	N	N	C++	400	15430	NA	38.58
MOD2	N	N	C++	240	11350	NA	47.29
Framework	NA	NA	Java	2720	9819	NA	3.61
MOD1	N	Y	Java	45	12439	0.85	276.42
MOD2	N	Y	Java	34	12231	0.87	359.74
MOD1v	Y	Y	Java	16	12865	0.97	804.06

는 시간당 코드 라인을 의미한다. 프레임워크를 사용하지 않고 C++ 언어를 사용해 개발한 경우는 MOD1 생산성이 38.58줄이고 MOD2는 47.29줄이다. 반면 자바 언어를 사용해서 프레임워크 자체를 개발하는 일은 시간당 3.61줄로 나타나고 있으며 이는 어플리케이션을 개발하는 것보다도 프레임워크 자체를 개발하는 것은 상당히 개발 생산성이 떨어지는 작업임을 알 수 있다.

이번에는 MOD1과 MOD2 개발하는데 프레임워크를 사용하는 경우의 개발 생산성을 측정하는 것으로서 MOD1이 프레임워크를 사용해 개발한 경우는 276.42 줄을 개발할 수 있으며 MOD2의 경우에는 359.74 줄을 개발 할 수 있다. 프레임워크를 사용하지 않는 경우보다 프레임워크 기반으로 어플리케이션을 개발하는 것이 개발 생산성면에서 우수하다. 하지만 재사용성이 가장 좋은 경우는 프레임워크를 특정 도메인에 맞게 수정한 후에 어플리케이션을 개발 할 경우로서 재 사용성이 97%에 이르고 있으며 LOC/Hour가무려 804.06줄의 코드를 작성할 수 있다.

CBD 어플리케이션 개발 시 개발 생산성을 향상시키기 위해서는 표 2의 결과에서 보듯이 프레임워크 기반으로 어플리케이션을 개발해야 하며 또한 특정 도메인에 맞도록 프레임워크를 재구성할 수 있는 유연하고 확장 가능한 아키텍처가 필요하다[18,20].

3. 소프트웨어 아키텍처

Morisio가 프레임워크 사용 여부와 가변성에 따른 생산성을 비교한 실험 결과에서 보듯이 생산성 향상을 위해서는 프레임워크 기반으로 개발이나 유지보수를 수행해야 하며 또한 특정 도메인에 맞도록 프레임워크를 재구성할 수 있는 유연하고 확장 가능한 아키텍처가 필요하다. 이러한 목적을 달성하기 위해 많은 프레임워크가 제안되었지만 프리젠테이션, 비즈니스 로직, 데이타베이스와 관련된 퍼시스턴트(Persistent) 단(Tier)에 걸친 통합적인 프레임워크를 제시하지 않아서 VSSH와 같은 각 단의 대표적인 프레임워크를 약결합 형태로 통합해서 사용하고 있다. 하지만 통합 자체도 어려울 뿐만 아니라 통합을 한다 해도 개발 툴킷 자체에 대한 통합이 어려우며 개발자가 사용하기가 어렵다는 문제점을 가지고 있다.

본 장에서는 CBD기반 어플리케이션 개발 시 생산성 및 유지보수 향상을 위해 유연하고 확장 가능한 품질 속성을 가진 아키텍처를 제시한다. 제시한 프레임워크는 VSSH 프레임워크와 같이 프리젠테이션, 비즈니스 로직, 퍼시스턴스와 같은 3개의 단(Tier)에 대한 프레임워크를 포함하고 있으며 더욱이 프리젠테이션 단과 비즈니스 로직단 프레임워크는 별도로 분리해서 사용할 수 있을 정도로 두개 단에 대한 프레임워크는 독립성을 유지할 수 있도록 설계되었다.

그림 4는 프리젠테이션과 비즈니스 로직 단을 포함한 전체 소프트웨어 아키텍처이다. 프리젠테이션단 프레임워크는 클라이언트 브라우저를 통해 요청을 받아 세션

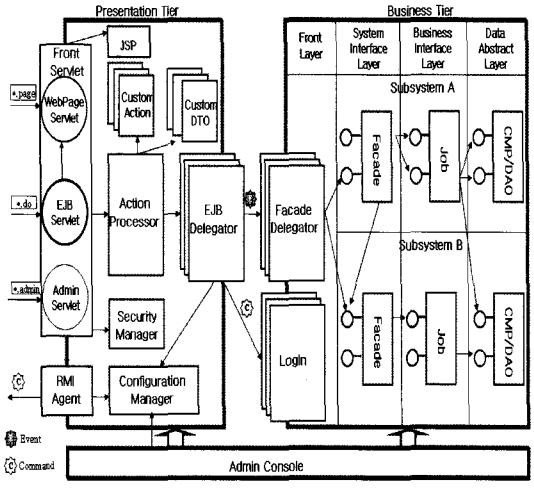

그림 4 프레임워크 전체 아키텍처

관리, 보안, 데이타포맷 변환을 처리하고 비즈니스 로직을 가지고 있는 EJB 모듈에 요청을 전송한다. 비즈니스 로직 프레임워크는 브라우저로부터 들어오는 클라이언트 요청을 처리하기 위한 비즈니스 로직을 가지고 있으며 어드민 콘솔은 유지보수 및 개발을 위한 툴킷들을 가지고 있다.

3.1 프리젠테이션 단 프레임워크

그림 4에서 제시한 전체 아키텍처 중에서 본절에서는 프리젠테이션 단 프레임워크에 대한 세부 아키텍처를 설명한다. 그림 5는 프리젠테이션 단의 아키텍처 구성도이다.

FrontServlet은 *.page, *.do, *.admin, *.login 같은 요청 URL패턴에 따라 클라이언트 요청을 처리하는 각각의 서블릿들이다. 예를 들면 *.do 패턴은 EJBServlet이 요청을 받아 비즈니스 로직을 처리하며 *.page 패턴은 PageServlet이 요청을 받아 어드민 콘솔에서 각 업무 구분 별로 설정한 페이지 구성 정보에 따라 화면 구성을 처리하는 작업을 수행한다.

FrontServlet이 처리하는 URL에는 업무 카테고리 정보가 있다. 예를 들어 클라이언트가 http://localhost/user.list.do와 같은*.do 형태의 URL 패턴을 사용했다면 어드민 콘솔상의 트리 계층 구조상의 업무 분류에는 user라는 업무 카테고리 이름이 존재하며 그 하위 트리에는 list라는 서브 업무 카테고리가 존재한다.

개발자나 유지보수 관리자는 어드민 콘솔상에서 트리 기반 계층 형태의 업무 카테고리별로 정책, 룰, 환경설정, 화면(예: JSP) 조합에 관련된 행위를 할 수 있으며 훅 메소드를 가진 Custom Action, DTO, EJB Delegator 그리고 비즈니스 로직을 가진 EJB 빈을 개발하고 컴파일, 패키징, 배포, 테스팅까지 수행할 수 있다. 또한 모든 자원이 트리계층 형태의 업무 카테고리별로 분류가 되어 있기 때문에 개발자나 유지보수자는 특정 자원에 대한 접근을 용이하게 할 수 있다.

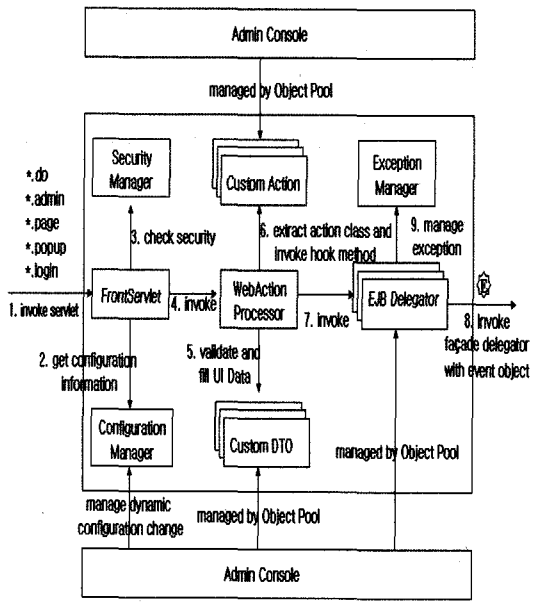

그림 5 프리젠테이션 단 아키텍처

ActionProcessor는 UI데이타를 CDTO(Custom DTO)에 채운 후에 액션 객체에 전송한다. 여기서 CDTO는 UI의 데이타를 담기 위한 객체이다. 액션객체는 Action-Processor에서 호출하는 훅 메소드를 가지고 있으며 EJB을 사용하지 않을 시 웹 단 비즈니스 로직을 구현한다. CDTO나 액션 객체는 요청을 처리하는 시점에 객체를 생성하지 않고 객체 풀에 이미 생성된 객체를 사용한다.

Configuration Manager는 어드민 콘솔에서 업무 카테고리 별로 지정된 XML기반 설정 정보를 관리한다. 어드민 콘솔상의 설정 정보가 변경되며 설정관리자가 관리하는 설정 정보도 동적으로 변경된다. FrontServlet는 클라이언트 요청을 처리하기 위해 필요한 여러 설정 정보들을 설정관리자를 통해 가져온다. EJBDelegator는 EJB빈과 통신을 담당하며 어드민 콘솔상에서 각 업무 카테고리 별로 구현, 컴파일 한 후에 객체 풀에 들어간다. EJBDelegator가 비즈니스 로직단으로 전송하는 객체는 그림 6과 같은 Event객체이다.

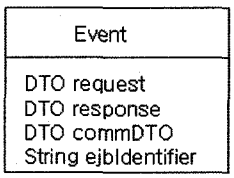

그림 6 Event객체

Event 객체의 Request는 UI에서 넘어온 값을 가지는 CDTO객체이고 Response는 EJB에서 비즈니스 로직 처리 후의 결과 값을 가지는 DDTO(Domain DTO)이며 CommonDTO는 비즈니스 로직 요청시 마다 필요한 공통정보를 담고 있는 객체이다. 문자열 타입의 ejbIdentifier는 비즈니스 로직 단에 있는 서브 시스템의 Facade빈을 식별하기 위한 구분자이다. Exception Manager는 비즈니스 로직단에서 예외가 발생한 경우 예외 처리를 담당한다.

EJB을 호출하고 난 후에 반환되는 결과 값은 DDTO에서 CDTO 형태로 반환되어 어드민 콘솔에 지정된 URL로 전송된다. 예를 들어 다음 URL 값이 *.page 패턴이면 EJB에서 반환된 데이타를 가지고 화면을 구성

하기 위해서 WebPageServlet으로 제어가 이동하며 이 때 해당 CDTO도 WebPageServlet으로 전송된다.

3.2 비즈니스 로직 단 프레임워크

전체 아키텍처 중에서 본절에서는 비즈니스 로직 단 프레임워크에 대한 아키텍처를 제시한다. 그림 7은 J2EE의 EJB 컴포넌트 형태로 비즈니스 로직을 구현하기 위한 아키텍처이다.

Facade Delegator는 모든 EJB 요청 진입점으로서 로그 처리, 예외처리 그리고 날짜 계산과 같은 모든 유스케이스들이 공통으로 처리해야 하는 작업을 수행한다. 프레임워크상에 모든 EJB 호출의 진입점 역할을 수행하는 한 개의 Facade Delegator에서 각 서브 시스템별로 존재하는 M개의 Facade Bean 사이에는 1: M 관계가 존재한다.

각 서브 시스템에 존재하는 Facade 빈을 호출하는 방식은 크게 두 가지로 분류할 수 있다. 첫번째 방식은 코더가Façade 빈을 호출하는 코드를 프레임워크에서 지정한 클래스의 훅 메소드에 구현한 후에 Façade Delegator와 함께 패키징(Packaging) 함으로써 Façade 빈을 호출하는 방식이다. 이 방식은 Facade빈의 인터페이스가 가지는 메소드를 호출하는 코드를 개발자가 작성해야 하는 방식이다. 두 번째는 Reflection 기법을 사용해 Facade 빈을 호출하는 방식으로 비즈니스 로직 작성자가 Facade 빈을 호출하는 코드를 작성할 필요가 없는 방식이다. Facade Delegator는 프리젠테이션 단에서 넘어온 Event 객체 안의 Façade 빈 식별자를 통해 각 업무 카테고리 별로 설정된 JNDI Name, Home Interface Name 등과 같은 정보를 얻어 Façade 빈을 호출한다.

Facade 빈은 컴포넌트 개발 프로세스 상의 분석 모델상의 각 유스케이스 별로 존재하는 시스템 인터페이스를 구현하는 EJB 빈이다. 주요 역할은 비즈니스 로직을 가지고 있는 빈을 호출하거나 다른 서브시스템의 비즈니스 로직과의 연동과 같은 비즈니스 프로세스를 조정하는 역할을 수행한다.

하나의 트랜잭션을 처리하기 위해 자신의 서브시스템이 가지는 비즈니스 로직만으로도 충분한 경우도 있지만 다른 서브시스템에 존재하는 비즈니스 로직을 필요로 하는 경우도 있다. 이런 경우 다른 서브시스템과의 모든 연동은 바로 Facade 빈에서만 수행한다. 또한 Facade 빈은 CDTO와 DDTO 사이의 데이타 변환을 수행한다. 비즈니스 로직에서는 DDTO만을 사용하고 프리젠테이션은 CDTO만을 사용하기 때문에 두 DTO사이의 변환에 대한 책임은 바로 Facade 빈에서 수행한다.

비즈니스 로직 빈은 비즈니스 도메인의 업무을 구현하며 Persistent 빈은 비즈니스 로직 빈과 관련해서 관계 형 데이타베이스나 XML 그리고 파일 시스템 등과

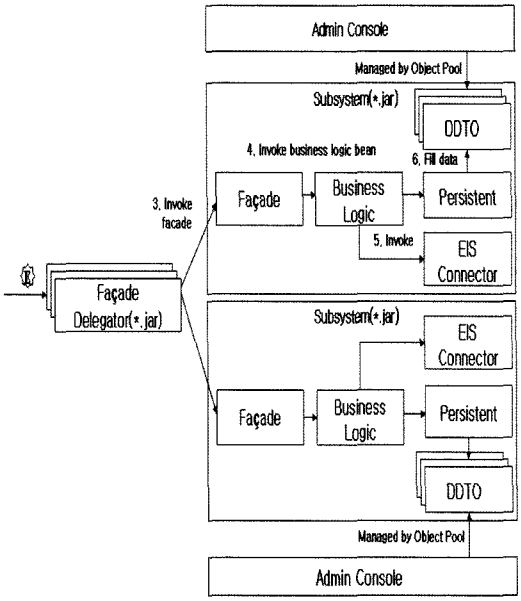

그림 7 비즈니스 로직 계층 아키텍처

관련된 비즈니스 데이타의 CRUD(Create, Read, Update, Delete)와 같은 기능을 제공한다. Persistent와 관련해 어드민 콘솔은 EJB Entity 빈을 위한 OR 매핑을 지원하며 이를 통해 코드 자동화 기능을 제공한다. 또한 DAO (Data Access Object)에 대해 SQL 쿼리 스크립트 기반 매핑 툴킷을 제공함으로써 DAO 코드 자동 생성 기능을 제공하기도 한다.

DDTO(Domain Data Transfer Object)는 EJB 비즈니스 로직 빈에서만 사용되며 단과 층 사이의 데이타를 전달하는 객체이다. EIS Connector는 SNA LU0, LU6.2 타입과 같은 레거시 시스템과 연동 시 사용되며 어드민 콘솔에서는 데이타 조합, 변환 그리고 모니터링을 위한 툴킷을 제공한다.

3.3 아키텍처 프로세스 처리과정

본 절에서는 클라이언트의 거래 요청에 대해 앞에서 설명한 아키텍처상의 주요 모듈을 사용해 거래를 처리하기 위한 아키텍처 프로세스 처리과정에 대한 설명이다. 그림 8은 제시한 프레임워크에 대한 동적 모델이다.

URL패턴에 매칭되는 FrontServlet이 요청을 받아 해당 업무 구분자를 식별한 후에 RequestFlow Manager를 통해 해당 설정정보를 가져오고 이를 기반으로 인증이나 접근 제어와 같은 전 처리 업무를 수행한다.

세션관리나 접근제어와 같은 전처리 업무를 수행 한 후에 ActionProcessor는 액션객체를 로딩해 훅 메소드를 실행한다. 그런 후에 EJB 비즈니스 로직 빈을 호출하기 위해 EJB Delegator를 호출한다. EJB Delegator

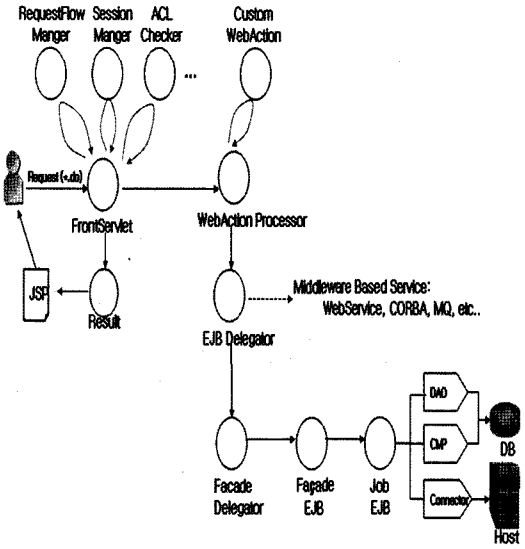

그림 8 프레임워크의 행위적인 프로세스 모델

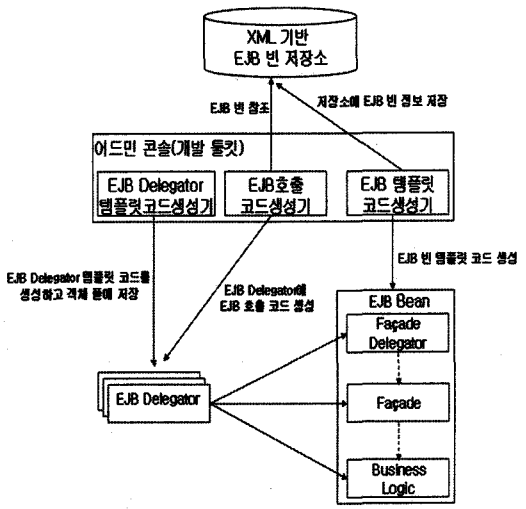

그림 9 주요 모듈 상호작용 변동 유연성

는 Event 객체를 생성한 후에 EJB요청의 관문역할을 수행하는 Facade Delegator를 호출한다. Facade Delegator는 날짜 계산이나 로그 처리와 같은 모든 유스케이스에서 필요로 하는 공통 기능들을 수행한 후에 Façade 빈을 호출한다.

Facade 빈은 비즈니스 로직을 가지고 있는Job빈을 호출하거나 다른 서브 시스템상에 존재하는 비즈니스 로직을 호출한다. Job빈은 비즈니스 로직을 가지고 있으며 비즈니스 로직을 처리하기 위해 필요한 데이타가 데이타베이스에 존재할 경우 CMP나 DAO를 사용해 처리한다. 만일 처리해야 할 데이타가 호스트에 존재한다면 EIS Connector를 사용해 가져온다.

3.4 유연성 달성 방안

3.4절에서는 개발 생산성과 유지보수 향상을 위한 유연성 품질속성 달성 방법들을 제시한다.

3.4.1 비즈니스로직단 업무별 처리 유연성

프레임워크에서 제공하는 기본적인 메시지의 흐름은 Facade Delegator를 통해 Facade 빈을 호출하고 Facade 빈은 비즈니스 로직 빈을 호출하는 것이다. 그림 9는 특정 업무 카테고리 에서 성능 품질 속성이 중요한 경우에 EJB Delegator에서 Facade Delegator를 호출하지 않고 직접 Facade빈이나 비즈니스 로직 빈을 호출할 수 있는 주요 모듈 상호작용 변동 유연성을 보여주고 있다.

어드민 콘솔상의 EJB 템플릿 코드 생성기는 EJB 빈 템플릿 코드를 생성하는 툴킷이다. 생성된 EJB 빈들은 EJB 빈 종류(예를 들면 EJB로컬 빈 혹은 리모트 빈)나 컴포넌트 타입 별(예를 들면 Facade 타입, 비즈니스 로직 타입)로 분류되어 XML 기반 EJB 빈 저장소에 저장된다. 개발자는 어드민 콘솔상에서 EJB Delegator 템플릿코드 생성기를 통해 EJB Delegator 템플릿 코드를 생성하고 EJB 호출코드 생성기를 통해 EJB Delegator 템플릿 코드 안에서 호출하고자 하는 EJB빈이나 메소드들을 선택함으로서 해당하는 빈의 메소드를 호출하는 코드가 작성된다. 이때 EJB 호출 코드 생성기는 XML 기반 EJB빈 저장소에서 EJB 빈을 참조해서 해당 EJB 빈과 메소드 리스트를 제공한다. 자동 생성된 EJB Delegator 코드는 어드민 툴킷을 통해 컴파일 되어 객체 풀로 들어가기 때문에 서버를 내리지 않고도 바로 테스트해 볼 수 있다.

각 업무 단위별로 성능이 중요한 경우와 확장성이나 유연성이 중요한 경우별로 비즈니스 로직 컴포넌트를 호출하는 것을 서로 다르게 선택해서 호출할 수 있다.

3.4.2 프리젠테이션 단에서 MVC의 유연한 동적 흐름 통제

본 논문에서 제시한 프레임워크는 전체적으로 MVC 아키텍처 패턴을 사용하고 있다. 그림 10은 어드민 콘솔을 사용해 MVC 패턴상의 컨트롤러, 뷰 그리고 모델간의 동적인 변경 가능성을 보여준다.

개발자나 유지보수 담당자는 어드민 콘솔상에서 업무 카테고리 별로 모델에 해당하는 EJB, 뷰에 해당하는 JSP, 컨트롤러에 해당하는 EJBServlet에 대한 흐름을 동적으로 변경될 수 있다. 예를 들어 클라이언트가 *.do패턴에 따라 요청을 보내고 프레임워크는 비즈니스 로직을 가진 EJB 빈을 처리한 후에 만일 어드민 콘솔상에서 그 업무에 해당하는 설정 내용중에 다음 URL이 *. page로 설정되었으며 WebPage Servlet으로 이동하

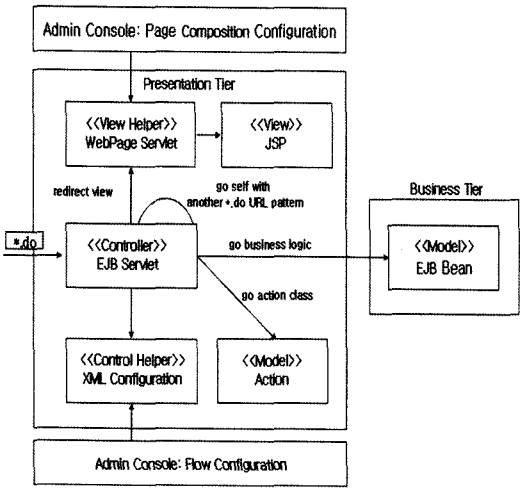

그림 10 어드민 콘솔을 사용한 유연한 MVC 통제

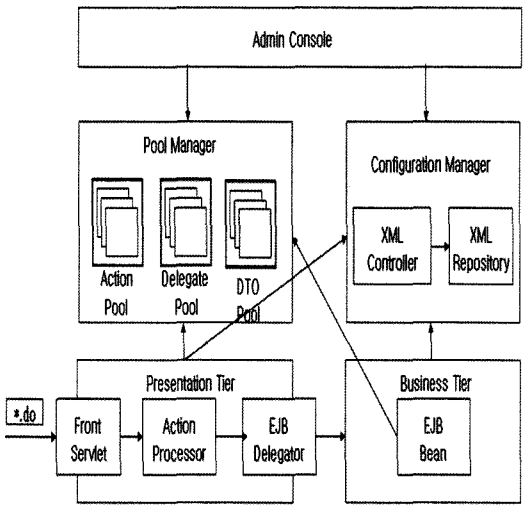

그림 11 객체 풀과 설정 매니저를 통한 유연한 유지보수 방법

며 *.do이면 다른 URL을 가지고 자기 자신인 EJB Servlet으로 이동하게 된다.

어드민 콘솔상의 각 업무 카테고리 별 페이지 구성 설정이나 다음 URL과 같은 설정 정보를 통해 개발자나 유지보수 담당자는 모델과 뷰 사이의 흐름 조정을 서버를 재가동하지 않고도 동적으로 변경할 수 있다.

3.4.3 동적기능 및 설정 변경 유연성

쉬운 유지보수를 위해서는 다양한 종류의 설정 정보, 관련 소스 등과 같은 방대한 양의 컨텐트에 대한 접근을 일관되고 효율적으로 할 수 있어야 한다. 또한 서버를 재 가동하지 않고 파라메타 설정, 소스코드 변경, 컴파일, 패키징, 분배 그리고 테스트를 할 수 있어야 한다. 그림 11은 프레임워크에서 제공하는 객체 풀과 설정 매니저를 통한 소스 코드 및 설정 정보의 동적인 변경 기법을 제시하고 있다.

개발자나 유지보수 담당자는 어드민 콘솔에서 계층형 트리 형태로 구분되는 업무 카테고리 별로 DTO, Action 그리고 EJB Delegator 코드를 개발하거나 수정하면 컴파일 되어 객체 풀로 들어가며 서버를 재 가동하지 않고도 새로이 개발된 코드를 바로 테스트해 볼 수 있다. 마찬가지로 룰이나 환경 설정 정보가 변경되면 Configuration Manager는 기존에 메모리에 로딩된 정보들을 버리고 새로운 정보를 로딩한다. 따라서 설정 정보 변경 시도 서버를 재 가동하지 않고 바로동적으로 반영할 수 있다.

방대한 양의 다양한 컨텐트에 대한 접근을 용이하게 하기 위해서 어드민 콘솔상에 계층 형 트리 형태로 업무를 구분하고 각 업무별로 다양한 타입의 리소스를 할당했다. 따라서 개발이나 유지보수 시 계층 형 트리에서 해당 업무 구분자를 찾아 지정된 리소스를 바로 접근해 리소스를 생성하거나 변경할 수 있으며 또한 변경 후에 서버를 내리지 않고도 바로 반영할 수 있기 때문에 개발이나 유지보수를 용이하게 할 수 있다.

3.4.4 비즈니스 컴포넌트 유연성을 위한 DTO 분리

비즈니스 컴포넌트의 재사용성과 유연성을 강화하기 위해서는 UI 화면상의 데이타 변동에 따라 비즈니스 로직상의 데이타 변동을 최소화해야 한다. UI데이타를 비즈니스 로직 컴포넌트에서 그대로 사용하는 경우 UI상의 데이타 변동이 생기면 바로 비즈니스 컴포넌트의 데이타도 변경해야 한다.

UI와 비즈니스 로직 컴포넌트 분리를 위해 그림 12와 같이 각 단과 계층 사이의 데이타를 이동하기 위해 사용되는 DTO객체를 CDTO와 DDTO로 분리해서 CDTO는 표현단과 비즈니스 로직 단의 Facade 빈까지 적용되고 그 이후의 비즈니스 로직 빈은 DDTO만을 사용한다. 이 두 DTO사이 변환은 Facade 빈에서 수행한다.

3.4.5 Facad 패턴을 사용한 비즈니스 로직 인터페이스의 유연한 변경

비즈니스 컴포넌트의 유연성과 재 사용성을 강화하기 위해서는 Facade패턴을 사용해 비즈니스 로직 컴포넌트가 가지는 인터페이스와 이를 호출하는 클라이언트간에 직접적인 의존관계를 없애야 한다. 또한 비즈니스 로직빈이 호출하는 다른 서브 시스템상의 비즈니스 로직빈과 의존 관계가 없도록 해야 한다. 이렇게 함으로서 비즈니스 로직빈의 인터페이스가 변경되었을 때 클라이언트의 파급효과를 최소화 할 수 있으며 비즈니스 로직빈이 호출하는 다른 서브 시스템상의 비즈니스 로직빈이 가지는 인터페이스가 변경될 경우 호출하는 비즈니스

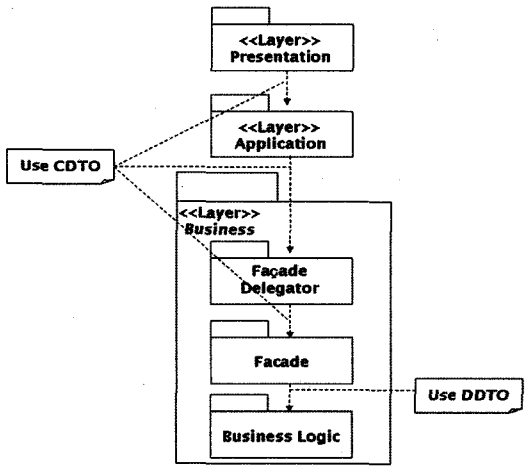

그림 12 UI와 비즈니스 로직 분리를 위한 CDTO와 DDTO분리

로직빈의 파급효과를 최소화 할 수 있다.

비즈니스 로직빈이 가지는 인터페이스와 클라이언트 간의 의존성을 최소화 하기위해 비즈니스 로직 컴포넌트가 가지는 비즈니스 인터페이스를 외부에 노출 시키지 않고 Facade 빈이 가지는 시스템 인터페이스만 외부에 노출 시킨다. 따라서 비즈니스 로직을 구현하고 있는 비즈니스 인터페이스가 변경되었을 때 Façade빈 컴포넌트 내부 구현에서 비즈니스 인터페이스를 호출하는 곳만 수정하면 되기 때문에 외부 클라이언트에는 영향을 미치지 않는다.

다른 서브 시스템상의 비즈니스 로직 빈 인터페이스 변경에 따른 파급 효과를 최소화하기 위해 다른 서브 시스템 상의 비즈니스 로직 컴포넌트에 대한 호출은 Facade 빈에서만 호출하도록 한다. 따라서 비즈니스 로직을 가지고 있는 비즈니스 로직빈에서 다른 서브시스템의 비즈니스 로직빈 인터페이스 변경에 따라 코드를 수정할 필요가 없다.

3.4.6 프레젠테이션과 비즈니스 로직 프레임워크의 독립성을 통한 개발 유연성

본 논문에서 제시한 프레임워크는 프리젠테이션 단과 비즈니스 로직 단 사이에 의존관계를 최소화 하도록 설계 되어 있어서 프리젠테이션단과 별도로 비즈니스 로직을 개발할 수 있다. 그림 13과 같이 프리젠테이션 단과 비즈니스 로직 단 사이의 의존관계는 EJB Delegator와 Façade Delegator에서만 존재하며 Facade Delegator에 Event 객체만을 전송하기 때문에 프리젠테이션단과 비즈니스 로직단을 별도로 개발하고 테스트하기가 쉽다.

또한 EJB 클라이언트로 본 논문에서 제시한 프리젠테이션단이 아닌 다른 제품들을 사용할 때에도 EJB Delegator만 추가함으로써 쉽게 연동 할 수 있다는 장점이 있다.

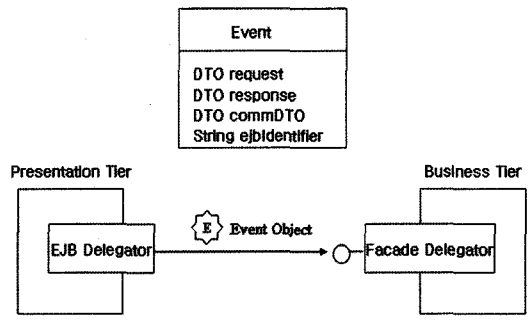

그림 13 프리젠테이션 단과 비즈니스 로직 단 의존성 최소화

3.5 확장성 달성 방안

본장에서는 제시한 프레임워크가 어떻게 기능 확장성 품질속성을 달성하는지에 대해 기술한다.

3.5.1 요청(Request)처리 확장성

프리젠테이션 단 요청과 비즈니스 로직 단 요청 처리에 대한 진입점은 각각 FrontServlet과 FaçadeDelegator이다. 만일 추가 기능이 존재 할 경우 어드민 콘솔상에서 각 업무카테고리별로 추가 기능을 훅메소드에 구현해서 진입점에 해당하는 FrontServlet이나 Facade-Delegator에 추가할 수 있다.

3.5.2 멀티채널 구현 확장성

비즈니스 로직 단이 제공하는 서비스를 호출하는 클라이언트는 다양한 프로토콜을 사용 할 수 있으며 또한 프레임워크와는 다른 데이타 포맷을 가질 수 있다. 그림 14는 다양한 프로토콜과 데이타 포맷을 사용하는 다양

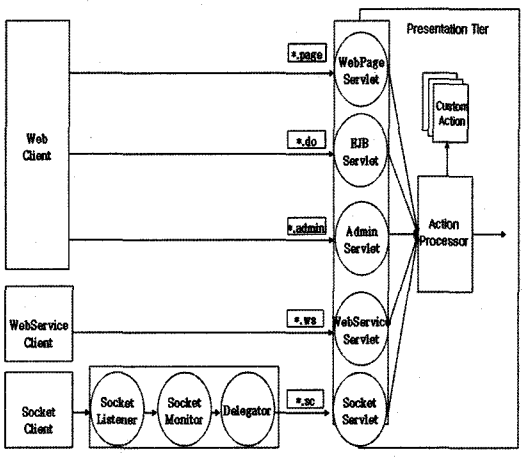

그림 14 멀티채널 구현 확장성

한 이기종 시스템과의 연동을 위한 멀티 채널 구현에 대한 확장성 설명이다.

본 논문에서 제시한 프레임워크는 TCP 소켓을 사용하는 클라이언트를 위해 Socket Listener을 만들었으며 웹 어플리케이션 초기화 시 구동된다. 소켓으로 들어오는 클라이언트에 대한 모니터링 기능을 제공하기 위해 Socket Monitor 객체가 필요하다. 다음으로 소켓으로부터 들어온 요청을 프리젠테이션 단 프레임워크에 전송하기 위해 Delegator를 통해 *.sc URL 패턴에 따라 SocketServlet으로 제어를 넘겨 처리한다.

웹 서비스 클라이언트의 경우는 HTTP 프로토콜에 SOAP 메시지가 전달되기 때문에 WebService Servlet를 통해 SOAP 메시지를 처리한다. 기타 다른 이기종 시스템과 연동 시 HTTP 프로토콜 기반 위에서 통신을 수행하는 경우에는 각 URL패턴에 해당하는 서블릿을 생성함으로써 처리하고 상이한 프로토콜로 인한 데몬(Daemon)이 필요한 경우에는 TCP 소켓과 같이 데몬을 통해 일단 요청을 받은 후에 다시 특정URL 패턴에 해당하는 서블릿을 만들어 처리한다.

3.5.3 멀티 인스턴스 환경에서 기능 확장성

최근 어플리케이션 개발이나 운영 환경은 어플리케이션 서버가 멀티 인스턴스인 경우가 많다. 그림 15는 웹 어플리케이션 서버가 멀티 인스턴스일 경우에 RMI Agent와 Command 패턴을 사용해 기능확장성을 달성하는 방법에 대해 설명하고 있다.

멀티 인스턴스 상황에서는 같은 이미지를 가지고 있어야 할 많은 어플리케이션 서버들이 존재하기 때문에 동기화를 유지할 필요가 있다. 이러한 환경에서는 싱글 사인 온(Single Sign On), 소스코드 동기화, 설정 동기화, 노드 자원 모니터링 그리고 버전 관리와 같은 많은 문제들을 해결해야 한다. 이런 종류의 기능 확장을 위해 각 어플리케이션 서버에 RMI Agent을 두고 있으며 기능 확장을 위해 RMI Agent간에 커맨드(Command) 패턴을 사용함으로써 멀티 인스턴스 환경에서 기능 확장을 용이하게 하고 있다.

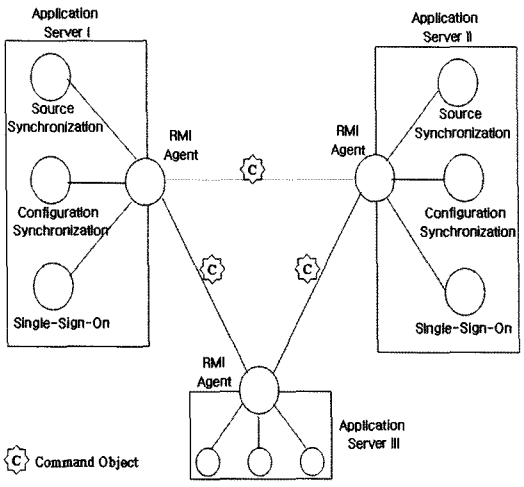

그림 15 RMI Agent와 Command 패턴을 사용한 기능 확장성

3.5.4 로그인 컴포넌트 기능 확장성

인증방법, 세션정보, 접근제어 정보에 대한 처리를 수행하는 로그인 컴포넌트는 매 프로젝트마다 상이하다. 본 논문에서는 EJB 로그인 컴포넌트에 Command 패턴을 사용해 기능을 쉽게 확장할 수 있도록 하고 있다. 따라서 프레임워크에서 제공하는 EJB 로그인 컴포넌트에 대해 수정이나 변경을 하지 않고 Command 객체에 필요한 코드만 추가하면 로그인 기능을 쉽게 확장할 수 있다.

4. 성능분석

본 장에서는 공개 소프트웨어이며 웹 어플리케이션 프레임워크 잘 알려진 Struts 프레임워크와의 성능을 비교한다.

4.1 실험환경

본 논문에서 제시한 프레임워크와 Struts 프레임워크와의 성능 비교를 위해서는 실험환경이 필요하다. 그림 16은 성능 실험분석을 위한 실험 환경을 나타내고 있다.

워크로드 생성과 결과 분석을 위해 웹 벤치 5.0을 사용했으며 웹 어플리케이션 서버로 웹 로직 6.1에 서비스 팩 7 버전을 사용했고 데이타베이스는 오라클 9i 버전을 사용했다. 또한 JBean은 프리젠테이션 뿐만 아니라 비즈니스 로직까지 포함하고 있으며 Struts와의 성능 비교 분석을 위해 프리젠테이션 단 부분에 대해서 만 성

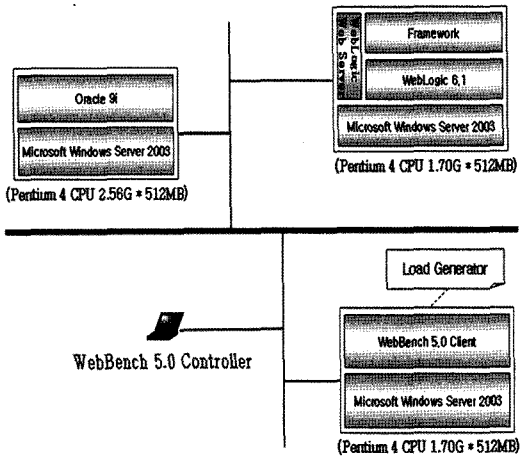

그림 16 성능실험 환경

능 분석을 수행했다.

표 3은 Struts 프레임워크와의 성능 비교를 위한 워크로드이다. 성능 비교를 위해 사용한 워크로드(Workload)는 웹 벤치 5.0에서 제시한 전자상거래용 어플리케이션을 테스팅하기 위해 제시한 워크로드 템플릿을 사용한다. 하지만 Struts와의 성능 비교를 수행하는데 필요없는 SSL, HTML 그리고 이미지를 제거하고 웹 어플리케이션과 관련있는 4개의 어플리케이션을 각각 가중치를 두어서 조합했으며 하나의 요청이 완료된 후에 다음 요청 시까지의 시간을 의미하는 Think Time을 3초로 잡았다. 4개의 어플리케이션 가중치와 Think Time은 일반적인 전자상거래 거래처리에 대한 로그 파일 분석을 통해 확인된 값이다. 표 3에서 Emp는 오라클의 Emp 테이블을 나타낸다.

표 3 성능실험을 위한 워크로드

No	Name	Tier	URL	Weight
1	로그인	프리젠테이션	GET/login.do	3(%)
2	Emp List	프리젠테이션	GET/emp.list.do	70(%)
3	Emp Insert	프리젠테이션	GET/emp.insert.do	10(%)
4	Emp Update	프리젠테이션	GET/emp.update.do	17(%)

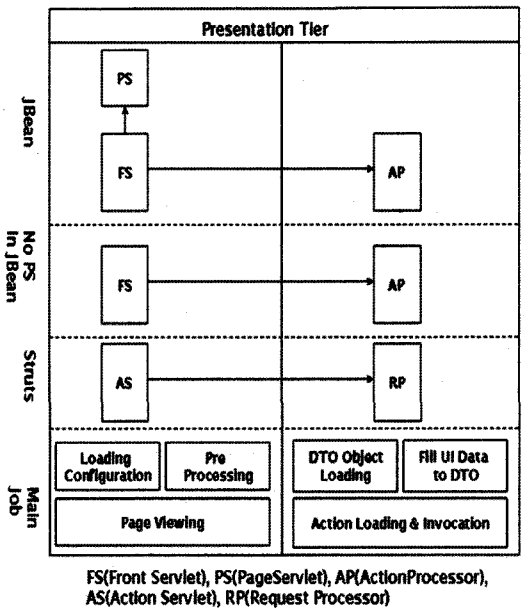

그림 17 Struts와 JBean 성능비교를 위한 3가지 아키텍처 모델

본 논문에서 제시한 프레임워크를 JBean이라 부르며 성능비교 및 분석을 위해 메티릭(Metrics)으로서 초당 트랜잭션 처리 건수를 나타내는 TPS(Transaction Per Seconds)를 사용하며 부하 분석을 위해서는 소프트웨어 아키텍처상의 주요 객체가 가지는 수행 시간을 사용한다.

4.2 Struts와 JBean의 프리젠테이션단 성능 비교

Struts 프레임워크는 프리젠테이션 단만 존재하며 본 논문에서 제시한 JBean 프레임워크는 프리젠테이션 단뿐만 아니라 EJB기반 비즈니스 로직 프레임워크까지 포함하고 있기 때문에 프리젠테이션 단에서만 Struts와 성능을 비교한다. 그림 17은 본 논문에서 제시한 프레임워크와 유사한 Struts 프레임워크와의 프리젠테이션 단만의 성능 결과를 비교하기 위한 3가지 아키텍처 모델과 주요 작업이다.

클라이언트 요청 URL 패턴이 *.do이면 표기상의 FS(FrontServlet)는 그림 4의 JBean전체 프레임워크 아키텍처상의 EJB Servlet에 해당하며 PS(PageServlet)은 WebPageServlet 해당된다. JBean과 Struts의 FS와 AS, AP와 RP는 비슷한 역할을 수행한다. FS와 AP는 요청 URL을 분석해서 업무 단위 별 XML 설정 정보를 가져오는 작업이며 다음으로 설정된 정보를 기반으로 인증, 접근제어 등과 같은 액션 수행 전에 처리해야 할 전 처리 작업을 수행하고 AP와 RP는 UI의 값을 저장하는 Struts상의 Form 객체와 JBean의 DTO 객체를 로딩해서 UI 데이타를 넣은 후에 Form 객체나 DTO 객체를 파라메타로 해서 액션 객체를 호출하는 것이다.

JBean과 Struts 프레임워크와 차이점은 Struts에서는 모델에 해당하는 액션 객체를 호출한 후에 지정된 JSP 페이지로 바로 이동하지만 JBean은 액션 객체 호출 후에 지정된 JSP페이지로 이동하는 것이 아닌 View Helper 패턴에 따라 페이지 프레임(Frame)에 지정된 JSP들을 조합하기 위해 PS를 거쳐 해당 JSP로 이동한다. 따라서 Struts와 비교해서 PS 만큼의 추가적인 작업들이 더 존재한다. 그림 18은 프리젠테이션 단에서 Struts와 JBean의 성능 비교 결과이다. Struts는 최대 TPS가 364이고 JBean에서 PS을 사용한 경우에는 최대 TPS가 322이고 JBean에서 PS를 사용하지 않는 경우 최대 TPS가 382이다. Struts와 같이 PS을 사용하지 않는 경우 JBean이 초당 18개 정도를 더 처리할 수 있다. 프리젠테이션단에 대해서만 Struts 프레임워크와의 성능 비교를 수행 했을 경우 Struts 프레임워크보다 더 우수한 성능 결과를 보였다. 하지만 JBean 프레임워크는 Struts에 없는 View Helper 패턴에 따른 PS(Page Servlet)를 추가적으로 가지고 있으며 또한 EJB를 기반으로 비즈니스 로직 컴포넌트를 만들 수 있는 비즈니스 로직단 프레임워크도 가지고 있다.

Struts에는 없는 PS와 비즈니스 로직 단 프레임워크까지 포함한 JBean 아키텍처가 가지는 주요 모듈을 다 포함할 경우 최대 220 TPS 정도의 성능 결과가 나온다. 그리고 두 프레임워크 주요 객체별 부하 분석결과

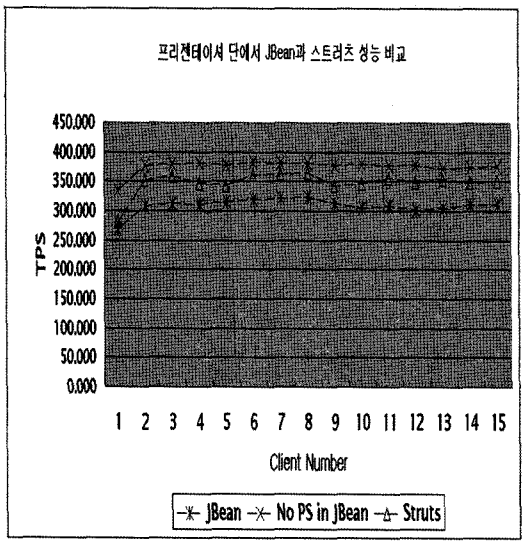

그림 18 프리젠테이션 단에서 JBean과 Struts의 성능 비교

Struts는 사용자 수가 증가함에 따라 Locale 처리와 같은 다국어 처리에 많은 수행 시간이 보이고 있다. JBean은 페이지를 구성하는 PS 부분에 가장 많은 수행 시간이 소여된다.

5. 결 론

최근 소프트웨어 개발 환경은 컴포넌트나 제품계열 방식과 같이 아키텍처 기반이며 비즈니스 전략이나 정책에 따라 가변적인 비즈니스 상황들이 많다. 이러한 상황에서 개발 생산성과 유지보수를 향상시키기 위해서는 재구성 가능한 유연하고 확장 가능한 아키텍처가 필요하다.

본 논문에서는 CBD기반 비즈니스 어플리케이션을 개발하기 위한 어플리케이션 프레임워크를 제시했다. 제시한 프레임워크는 유연하고 확장 가능한 아키텍처를 가지고 있다. 유연성을 달성하기 위해 프레임워크 아키텍처는 비즈니스 로직단의 업무별 처리 유연성, 프리젠테이션단의 업무별 처리 프로세스 변경 유연성, 동적 기능 및 설정 변경, UI와 비즈니스 로직 분리성, 프리젠테이션과 비즈니스 로직단 프레임워크 개별적 분리, 비즈니스 로직 인터페이스의 유연한 변경과 같은 것들을 해결하기 위해 설계되었다. 또한 본 논문에서 제시한 프레임워크는 요청 처리 확장성, 멀티채널 구현 확장성, 웹 어플리케이션 서버 멀티 인스턴스 환경에서 기능적인 확장, 로그인 컴포넌트 확장과 같은 확장성 품질을 달성하고 있다.

본 논문에서는 또한 엔터프라이즈 어플리케이션 개발시 일반적으로 많이 사용되는 Struts 프레임워크와의 성능 비교 결과를 수행했다. Struts에서는 존재하지 않는 PS라 불리는 WebPageServlet을 사용하지 않는 상태에서 본 논문에서 제시한 프레임워크는 Struts 프레임워크보다 18 TPS만큼 더 많은 처리량을 보이고 있다.

참 고 문 헌

[1] F. Barbier. Component-based design of large-scale distributed systems. In Proc. COMPSAC, pages 19 - 24, 2001.

[2] J. X. Ci and W. T. Tsai. Distributed component hub for reusable software components management. In Proc. COMPSAC, pages 429 - 435, 2000.

[3] D'souza D.F. and Wills A.C., Objects, Components, and Components with UML, Addison-Wisely, 1998.

[4] Weiss, David M., Lai, Chi Tau Robert. 1999. Software Product-line Engineering. Addison-Wesley, Reading, MA.

[5] M. Morisio and C. B. Seaman et al, Investigating and improving a COTS-based software development process, ICSE 2000, pp.31-40, 2000.

[6] Lars Geyer and Martin Becker, "On the influence of Varaibilities on the Application-Engineering Process of a Product Family," Proceedings of SPLC2, 2002.

[7] Herbert Weber, Asuman Sunbul and Julia Padberg, Evolutionary Development of Business Process Centered Architectures Using Component Technologies, Technical University Berin.

[8] J. Kramer and J. Magee, Analyzing dynamic change in distributed software architectures, IEE Proceedings-Software, 145(5), Oct. 1998.

[9] Jeong Ah Kim, YoungTaek Jin, SunMyung Hwang: A Business Component Approach for Supporting the Variability of the Business Strategies and Rules. ICCSA (3) 2005: 846-857

[10] Fumihiko Kitayama, Shin-ichi Hirose, Goh Kondoh, Design of a Framework for dynamic Content Adaptation to Web-Enabled Terminals and Enterprise Applications, IEEE, 1999.

[11] M. Fayad, D, Schemidt, and R. Johnson, eds., Building Application Framework, Wiley & Sons, New York, 1999.

[12] Apache, Struts Framework, http://jakarta.apache.org/struts/index.html

[13] Apache, Velocity Framework, http://jakarta.apache.org/velocity/

[14] Spring Framework, http://www.springframework.org/

[15] Sun MVC Information at Sun Micro System Web Site, http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html

[16] Sun Microsystems: Enterprise JavaBeans 2.0 Final Specification, September 2001.
[17] Maurizio Morisio, Carolyn B. Seaman, Victor R. Basili, Amy T. Parra, Steve E. Kraft, Steven E. Condon: COTS-based software development: Processes and open issues. Journal of Systems and Software 61(3): 189-199 (2002)
[18] Maurizio Morisio, Ioannis Stamelos, Vasilis Spahos, Daniele Romano: Measuring Functionality and Productivity in Web-Based Applications: A Case Study. IEEE METRICS 1999: 111-118.
[19] Maurizio Morisio, Daniele Romano, Corrado Moiso: Framework Based Software Development: Investigating the Learning Effect. IEEE METRICS 1999: 260-268.
[20] Maurizio Morisio, Daniele Romano, Corrado Moiso: Framework Based Software Development: Investigating the Learning Effect. IEEE METRICS 1999: 260-268.

이 용 환

1997년 건국대학교 행정학사. 1999년 건국대학교 공학석사. 2006년 건국대학교 공학박사. 2005년 (주) 인터넷 커머스 코리아 연구소장과 동덕여대 겸임교수. 2006년 현재 건국대학교 연구교수로 재임